Green University of Bangladesh
Department of Computer Science and Engineering (CSE)
Faculty of Sciences and Engineering
Semester: (Fall , Year:2025), B.Sc. in CSE (Day)

Lab Report NO: 03
Course Title: Artificial Intelligence lab
Course Code: CSE 316 Section:231 D3

Lab Experiment Name: Alpha Beta pruning

Student Details

	Name	ID
1.	Promod Chandra Das	231002005

Lab Date : 10.12.25
Submission Date : 17.12.25
Course Teacher's Name : Mr.Mozdaher Abdul Quader

Lab Report Status

Marks: …… **Signature:.....**
Comments:..... **Date:.....**

1.TITLE OF THE LAB REPORT EXPERIMENT

Alpha Beta Pruning

2. OBJECTIVES

The objective of this experiment is to study and implement the Alpha–Beta Pruning technique used in game tree search algorithms. This experiment aims to understand how Alpha–Beta Pruning improves the efficiency of the Minimax algorithm by eliminating unnecessary branches in the game tree without affecting the final decision. It also helps in understanding adversarial search and optimization in decision-making problems.

3.PROCEDURE

Initially, a game tree representing possible moves of two players is considered, where one player acts as the maximizer and the other as the minimizer. The Minimax algorithm is applied to evaluate the game tree by recursively exploring all possible moves and assigning utility values to terminal nodes. During this process, two values known as alpha and beta are maintained. The alpha value represents the best value that the maximizer can guarantee so far, while the beta value represents the best value that the minimizer can guarantee so far. As the tree is explored, alpha and beta values are updated accordingly. If at any point the alpha value becomes greater than or equal to the beta value, further exploration of that branch is stopped, as it cannot influence the final decision. This pruning process continues until the optimal move is determined.

4.IMPLEMENTATION

4.1

7 Lab Exercise (Submit as a report)

Implement a simple Tic-Tac-Toe game between a human player and an AI agent. The AI should use the Minimax algorithm with Alpha-Beta Pruning to choose the best move. Print how many nodes were pruned during each AI turn.

7.1 Input

Input of the program is given below.

```
1,0,1
0,1,x
x,x,0
```

7.2 Output

Output of the program is given below.

```
Nodes evaluated: 47
Nodes pruned: 22
```

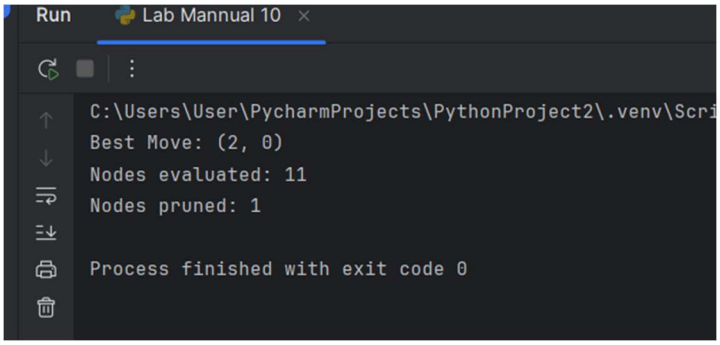
Answer:

```
AI = 1
HUMAN = 0
EMPTY = 'x'

nodes_evaluated = 0
nodes_pruned = 0


def is_moves_left(board):
    for row in board:
        if EMPTY in row:
            return True
    return False


def evaluate(board):
    for i in range(3):
        if board[i][0] == board[i][1] == board[i][2] != EMPTY:
            return 10 if board[i][0] == AI else -10
        if board[0][i] == board[1][i] == board[2][i] != EMPTY:
            return 10 if board[0][i] == AI else -10

    if board[0][0] == board[1][1] == board[2][2] != EMPTY:
        return 10 if board[0][0] == AI else -10
    if board[0][2] == board[1][1] == board[2][0] != EMPTY:
        return 10 if board[0][2] == AI else -10

    return 0


def minimax(board, depth, alpha, beta, is_max):
    global nodes_evaluated, nodes_pruned
    nodes_evaluated += 1

    score = evaluate(board)

    if score == 10 or score == -10:
        return score
    if not is_moves_left(board):
        return 0

    if is_max:
        best = -1000
        for i in range(3):
            for j in range(3):
                if board[i][j] == EMPTY:
                    board[i][j] = AI
                    best = max(best, minimax(board, depth + 1, alpha, beta, False))
                    board[i][j] = EMPTY
                    alpha = max(alpha, best)
                    if beta <= alpha:
                        nodes_pruned += 1
                        return best
```

```
        return best
    else:
        best = 1000
        for i in range(3):
            for j in range(3):
                if board[i][j] == EMPTY:
                    board[i][j] = HUMAN
                    best = min(best, minimax(board, depth + 1, alpha, beta, True))
                    board[i][j] = EMPTY
                    beta = min(beta, best)
                    if beta <= alpha:
                        nodes_pruned += 1
                        return best
        return best


def find_best_move(board):
    best_val = -1000
    best_move = (-1, -1)

    for i in range(3):
        for j in range(3):
            if board[i][j] == EMPTY:
                board[i][j] = AI
                move_val = minimax(board, 0, -1000, 1000, False)
                board[i][j] = EMPTY
                if move_val > best_val:
                    best_val = move_val
                    best_move = (i, j)
    return best_move


# Input board (as per lab question)
board = [
    [1, 0, 1],
    [0, 1, EMPTY],
    [EMPTY, EMPTY, 0]
]

best_move = find_best_move(board)
print("Best Move:", best_move)
print("Nodes evaluated:", nodes_evaluated)
print("Nodes pruned:", nodes_pruned)
```

5.TEST RESULT / OUTPUT

4 .1 : OUTPUT



Fiqure: Result/ Output 4.1

6.ANALYSIS AND DISCUSSION

Alpha–Beta Pruning significantly improves the performance of the Minimax algorithm by reducing the number of nodes that need to be evaluated. While the basic Minimax algorithm explores all nodes in the game tree, Alpha–Beta Pruning avoids exploring branches that do not affect the final outcome. In the best case, Alpha–Beta Pruning reduces the time complexity from O(b^d) to O(b^(d/2)), where b is the branching factor and d is the depth of the tree. This allows deeper game trees to be searched within the same time constraints.

However, the effectiveness of Alpha–Beta Pruning depends heavily on the order in which nodes are evaluated. Good move ordering results in more pruning, while poor ordering reduces its efficiency. Despite this limitation, Alpha–Beta Pruning guarantees the same optimal result as the Minimax algorithm and is widely used in game-playing artificial intelligence systems due to its correctness and improved efficiency.

7. SUMMARY

In this experiment, the Alpha–Beta Pruning technique was studied as an optimization of the Minimax algorithm used in adversarial game playing. By maintaining alpha and beta values during the search process, the algorithm successfully eliminated branches of the game tree that did not affect the final decision. This reduced the number of nodes evaluated while still guaranteeing the same optimal result as Minimax. The experiment demonstrated that Alpha–Beta Pruning improves efficiency and allows deeper game trees to be explored, making it an effective and widely used technique in artificial intelligence applications.